A Lightweight Dynamic Storage Algorithm with Adaptive Encoding for Energy Internet

Song Deng, *Member, IEEE*, Yujia Zhai, Di Wu, *Member, IEEE*, Dong Yue, *Fellow, IEEE*, Xiong Fu, Yi He, *Member, IEEE*

Abstract—Reliable data storage is crucial to the production, transmission, transaction, consumption, and analysis of an Energy Internet (EI). Whereas mainstream distributed data storage seems to be a plausible solution, the existing methods suffer from a tradeoff between the storage overhead (incurred by the replicas of data encodings for lossless recovery) and the communication latency (due to the spiking network traffic resulting from massive queries of data replicas across devices). To balance this tradeoff, we propose a *Lightweight Dynamic Storage Algorithm based on Adaptive Encoding* (LDSA-AE) approach for EI data storage. Our key idea is to classify the data into active and inactive categories, where the active data are most likely to be accessed and thus corrupted in high frequencies. As such, wherever the active data are housed, the replicas of them can be proactively allocated into a set of nearby devices. The main challenges are to realize the classification in real-time and to tailor encoding methods for the active and inactive separately in correspondence to their own characteristics. To overcome these, our LDSA-AE 1) proposes a novel density-based clustering algorithm to tackle performance data classification in an online and unsupervised fashion and 2) leverages Minimum Density RAID-6 (MDR) code and Cauchy Reed-Solomon (CRS) code for active and inactive data encodings, respectively, striving to ensure data storage with low overhead, low latency, high reliability, and high throughput at once. A theoretical analysis substantiates the viability and effectiveness of our proposed LDSA-AE approach. We also prototype our LDSA-AE on a real-world server testbed, and the empirical study suggests the superiority of our approach over the state-of-the-art distributed storage schemes for EI in terms of storage overhead, repair throughput, and reliability.

Index Terms—Erasure code, Density clustering, Minimal density RAID-6 code, Adaptive scheme, Storage system, Energy Internet.

NOMENCLATURE

AE	Adaptive Encoding
CDM	Coding Distribution Matrix
CRS	Cauchy Reed-Solomon
DCUACA	Density Clustering-Based Automatic Unsupervised Classification Algorithm
EC	Erasure Code
EI	Energy Internet
FS	File System
HDFS	Hadoop Distributed File System
HH	Hitchhiker
HRepair	Hybrid-Structured Repair Method of Cross-Datacenter Erasure Code
LDSA-AE	Lightweight Dynamic Storage Algorithm with Adaptive Encoding
LRC	Locally Repairable Code
MDR	Minimum Density RAID-6
MDS	Maximum Distance Separable
MTBF	Mean Time Between Failures
NEC	Network Environment-Adaptive Encoding Method of Cross-Datacenter Erasure Code
RAID-6	Redundant Arrays of Independent Drives-6
RS	Reed-Solomon
XOR	Exclusive OR

Manuscript received XXXX XX, 2022; revised XXXX XX, 2022; accepted XXXX XX, 2022. This work was supported by the National Natural Science Foundation of China under Grant No. 51977113, 62293500, 62293501, 62293505, and 62176070. Postgraduate Research & Practice Innovation Program of Jiangsu Province under Grant No. SJCX21_0291, and BAGUI Scholar Program of Guangxi Zhuang Autonomous Region of China under Grant No. 201979. Corresponding Authors: Di Wu, wudi.cigit@gmail.com

Song Deng is Institute of Advanced Technology, Nanjing University of Posts & Telecommunications, Nanjing 210003, China (e-mail : dengsong@njupt.edu.cn)
Yujia Zhai, Dong Yue are with the College of Automation, Nanjing University of Posts and Telecommunications, Nanjing, China (e-mail: zhaiyujia9805@163.com, yued@njupt.edu.cn).
Di Wu is with the College of Computer and Information Science, Southwest University, Chongqing 400715, China, (email: wudi.cigit@gmail.com)
Xiong Fu is with the School of Computer, Nanjing University of Posts & Telecommunications, Nanjing 210003, China (e-mail : fux@njupt.edu.cn)
Yi He is with Old Dominion University, Norfolk, Virginia 23462, USA (e-mail : yihe@cs.odu.edu)

1 INTRODUCTION

ENERGY Internet (EI) has emerged to harmonize energy flow, information flow, and business flow, creating an inclusive and efficient realization of smart energy system [1]. A typical EI often needs to support the smooth operation of millions of devices and business systems, including a diversity of distributed energy generators (e.g., microturbines and fuel cells), accumulators (e.g., batteries), and consumers (e.g., electric vehicles), which are driven by multiple energy modalities (e.g., electricity, natural gas, photovoltaic power). A schematic view of EI is illustrated in the left panel of Figure 1.

Despite its promising industrial potential, the EI system mainly suffers a complex and interleaved data interconnection. Indeed, the data generated from and communicating across the infrastructures and devices are the key tone to enable the production, transmission, transaction, consumption, and analysis in the EI systems. Unfortunately, the data

pervading all corners of EI tend to invite various cyber attacks [2], [3], eventually incurring exacerbated data damage in storage.

To wit, two real cases of data storage attacks toward EI are exemplified as follows. *Case I (Ukraine, 2015):* Three power distribution companies in Ukraine were attacked, causing power outages to more than 80,000 residents [4]. The adversary injects data between business and control system networks through malware via phishing emails. *Case II (United States, 2018):* A denial-of-service (DoS) attack on unpatched firewalls was performed on the control room of the U.S. utility, resulting in the loss of visibility of 500 megawatts (MW) of power generation assets in the U.S. [5]. The attacked power companies experienced intermittent service due to frequent firewall restarts. All data services were implicated, and damaged data could not be backed up and restored in a timely manner.

Urged by the situation, how to enable secure, accountable, and economically efficient data storage in the EI systems has drawn extensive attention from academia and industry. A common practice to this end is to simply store multiple replicas of the data in an external database. Once the system shut down due to data corruption, we can recover the damaged data from their corresponding replicas, with the crux lying in the efficient probing and query of the data corruption and their replicas. However, the massive amount of data generated from EI constantly makes the *scalability* of this so-called "Replication Strategy [6]" questionable – the entire storage space needed by the EI data grows in a multiplicative fashion and can soon become unmanageably large over time.

In response, one may think to adapt the distributed storage system to resolve the scalability issue, such as Microsoft Azure [7], Meta cloud warehouse [8], Aliyun [9], Hadoop Distributed File System [10], and Ceph [11], to name a few. There are two main procedures of such distributed storage systems. First, the original data are represented by a new encoding system in an *over-complete* manner [12], in the sense that a *subset* of the new encoding is able to *recover* the original data in a lossless fashion. Second, instead of using an external database, the replicas of the new data encodings are stored in the devices across the EI in a distributed manner. As such, once the data of any device corrupt, we inquiry its neighboring devices to provide recovery replicas. Seemingly, the larger number of devices served by the EI, the more distributed storage can be harvested to enable a damage-resilience storage system, which is hence scalable.

Unfortunately, this distributed storage solution may not be adapted to an EI context so perfectly. Indeed, the heterogeneous, multi-modal, and large-scaled natures of data abounding in EI systems tend to incur *a tradeoff between storage overhead and communication latency* in a distributed regime. Consider, for example, securing those data being high-dimensioned (e.g., images, audio data, and videos), whose over-complete encodings would consume a considerably massive amount of memory resources. The tradeoff arises where deciding the number of replicas of the data encodings needing to be stored is close to impossible in ad-hoc. Specifically, a small number of replicas relieves the storage burden, while it triggers the spiking communication traffic as one corrupted device has to revoke a large number of other devices holding its replicas. Network congestion is most likely to occur, and data corruption which often happens just randomly eliminates a pre-scheduling. On the opposite, a large number of replicas may mitigate the communication latency, as the probability that a corrupted device may find a recovery replica from its nearby devices or even on its own is substantially higher. Alas, such high-dimensional data replicas lead to storage redundancy, and increase the disk I/O and read degradation for normal data, which lowers the efficiency of data query in EI systems.

Motivated by this tradeoff, we in this paper mainly explore one question – *Can we build a distributed and secure data storage system for Energy Internet without being bothered by the storage and communication bottlenecks?*

An affirmative answer provides us a novel approach, termed *Lightweight Dynamic Storage Algorithm based on Adaptive Encoding* (LDSA-AE). Our key observation that motivates the LDSA-AE design is that the data accesses from the user end to the EI devices often follow long-tailed distributions. As reported by [13], the frequencies of data access for production Hadoop clusters at Meta and four different Cloudera customers exhibit a Zipf distribution, where most access requests converge to a small data proportion, while the vast majority of the remaining data are accessed very rarely. An optimal balance between the storage and communication for data recovery can thus be envisioned by a dynamic classification of "active data" (being frequently accessed) and "inactive data" (being rarely queried). Intuitively, by preparing a larger number of replicas at the nearby devices for the active data only, a higher data recovery efficiency can be attained without a substantial increase in storage and communication requests.

Along with this observation, the main challenges are two-fold as follows. 1) Realizing the active/inactive data classification which must be performed in a *real-time* fashion. A delayed code switch from inactive to active data is most likely to incur data damage [14], and 2) Tailoring proper over-complete encoding methods for the active and inactive data separately to maximize the hit ratio of recovery replicas. As the storage occupied by the inactive data overwhelms that by the active data, heuristic methods that exhaust the entire storage space [15] fail to work well. Our LDSA-AE approach strives to overcome the two challenges. More specifically,

Our major contributions are summarized as follows:

1) A density-based clustering algorithm is proposed to enable the active/inactive data classification in a real-time and unsupervised manner. The clustering standard of data activity is defined, and the data storage scheme is dynamically and adaptively switched according to the workload and data characteristics to ensure the lightweight and efficiency of the whole EI system.
2) Minimum Density RAID-6 (MDR) code, along with the Replication strategy, are employed for active data encoding in Energy Internet, which improves the recovery throughput and reliability of active data and reduces degraded read delay and storage overhead.
3) Cauchy Reed-Solomon (CRS) code is proposed for the inactive data storage scheme in EI, which con-

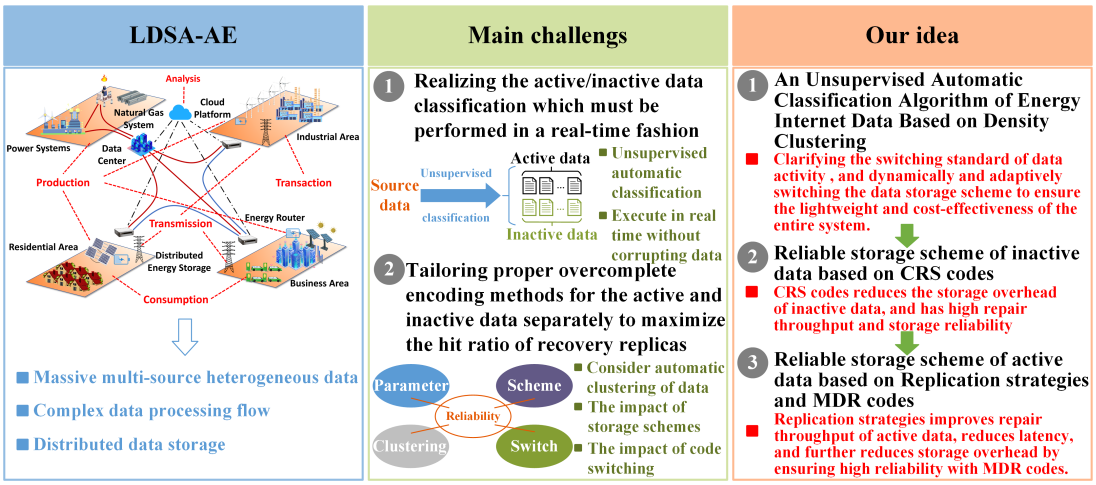

Fig. 1. Energy Internet background, the three challenges, and our main ideas to tailor LDSA-AE.

siderably reduces the storage cost of inactive data and improves the repair throughput and storage reliability.

4) A theoretical analysis is carried out, and the results substantiate the viability of the effectiveness of our proposed LDSA-AE approach.
5) We prototype our LDSA-AE solution on a real-world server testbed and evaluate its performance over a diversity of benchmarks. A comparative study suggests that our LDSA-AE significantly outperforms the state-of-the-art distributed data storage solutions for EI in terms of higher reliability, less storage overhead, and larger communication throughput.

The rest of this paper is organized as follows. Section 2 presents the related work. We scrutinize the LDSA-AE architecture and design in Section 3, and analyze its theoretical properties in Section 4. Section 5 documents the evaluation results. We conclude this paper in Section 6.

2 RELATED WORK

To date, the existing data storage solutions for Energy Internet are mainly divided into two categories, namely, a single coding scheme and a combined coding scheme. This section serves to discuss the pros and cons of these schemes and their relations to our proposal in this paper.

2.1 Single encoding scheme

In general, most Erasure Code (EC) systems use a single erasure coding. For example, both Google and Facebook have adopted RS codes while azure LRC [16]. Diverse choices are made because the distinctions in code families and parameters greatly affect a storage system in various aspects, including reliability, performance, and even structure. With the passage of time, its own disadvantages are gradually reflected. [17] through the improvement of Replication strategy and RS code, the system has achieved higher flexibility while reducing storage and network costs. The former proposed a new craft protocol based on the raft protocol and provided two different Replication methods. The latter adopted a hierarchical storage strategy, which used Replication strategy and EC to store different types of data respectively. At the same time, the number of data blocks and parity blocks can be changed according to the workload, so as to achieve an optimal balance. Some studies have sought to solve the disadvantages of RS code through LRC [16], by LRC encoding every $k/2$ data in a stripe while storing parity blocks in a new storage device. This solution is widely used by Microsoft azure [7]. An efficient fault recovery algorithm of D^3 defined in [18]. The orthogonal array is used to evenly place data blocks and parity blocks, which significantly improves the fault recovery speed of RS code by 2.49 times and that of LRC by 1.38 times. However, LRC is not an MDS code. The recovery speed has been improved, but the delay has become higher. Therefore, there will be a trade-off between storage overhead and recovery delay.

Efficient storage schemes based on RAID-6 level are [19], they have proposed efficient RAID-6 scaling schemes on the basis of the original algorithms. By optimizing data migration and parity data update, the total cost of parity data update is finally reduced and the total scaling time is saved. Vectorization XOR operation may be a better choice than direct vectorization finite field operation [20]. It can be seen that in most cases, a single encoding scheme cannot minimize the recovery cost while keeping the storage overhead low, and it is difficult to get rid of the inherent defects of the code family. Especially for the massive multi-source heterogeneous data of the Energy Internet, although the complexity of the entire system is reduced and the management convenience is optimized, a trade-off must be made between overhead and performance, so a single coding scheme is not in line with the application environment of Energy Internet.

2.2 Combined encoding scheme

Dynamic redundancy control is an effective solution, it can select the best storage solution according to workload and data type, utilize different storage algorithms, retain the advantages of each algorithm, complement the disadvantages, and achieve high storage efficiency without losing too much computational performance.

In [21], in order to reduce the network bandwidth in the cloud storage system, appropriate recovery techniques are defined in combination with Replications, ECs, and

delayed recovery methods to effectively reduce the latency of degraded reads in cloud storage. However, user applications need to access data and file metadata remotely, which will inevitably lead to an increase in network overhead and an unavoidable impact on performance. Recent studies have proposed some EC storage schemes using efficient code-switching algorithms, [14] used LRC to store frequently accessed data, cooperated with HH to store infrequently accessed data, and combined LRC and HH to develop a code switching algorithm, which guarantees low latency and low network overhead of the storage system, and is not limited by code-switching. In addition to using multiple algorithms to develop adaptive storage schemes, [22] proposed a network adaptive coding method, NEC, which can generate matrices online through approximately optimal parallel heuristics to achieve approximately minimal average weighted locality of data placement schemes. Then, an HRepair hybrid structure repair method was proposed to further reduce the maintenance cost and maintenance time. Unfortunately, these schemes have some drawbacks, with bandwidth consumption issues during code-switching. In a word, in the process of data storage, the control of data storage overhead, repair efficiency, and data reliability is an urgent problem to be solved in the existing industrial storage environment.

Discussion and Our Idea: Data corruption is a significant hidden danger to Energy Internet security. It may occur on any server or disk in the Energy Internet. Once these servers or disks fail, user applications cannot receive user files in time, and even trigger cascading failures. The high degree of fragmentation of Energy Internet data makes the occurrence of data loss more widespread. In addition to data loss, security risks include network bandwidth environment and data recovery measures. However, most of the above data storage strategies are based on storage overhead and system performance and do not consider the reliability of encoding storage. Moreover, the storage schemes in the above-mentioned documents neither accurately reveal the relationship between storage overhead, repair throughput, and code parameters, nor consider the impact of storage schemes on reliability. Therefore, adaptive coding requires trade-offs when choosing a code. **The summary of the comparisons among different code families is shown in S1 in the Supplementary File of this paper, which can be downloaded from https://github.com/dsylc2006/Supplementary_documents_TSC-2022-04-0146__R1.**

Across the three code families, changes in parameters lead to changes in performance. In this paper, we consider storage overhead, recovery throughput, reliability, and read latency, which is commonly considered by storage systems. Let k be the number of data blocks, m be the number of parity blocks, w be the bit size, packet size is the size of each data packet, and buffer size is the data buffer size. Among them, we can use m/k to represent the storage overhead of data redundancy, k and m respectively reflect the recovery cost and fault tolerance capacity of the storage system. For performance reasons, we would like to see small m/k, small k, and large m. However, while only ideally possible, it is actually quite difficult to implement them completely, and these three conditions are always mutually exclusive. System reliability is related to the use of different storage strategies, and the repair throughput of the system will inevitably be affected by high reliability.

The combination selection of different code families and parameters is a key factor for us to achieve the target performance. As discussed in the section, each code composition scheme necessarily has its own pros and cons. Using only one code scheme to store data across the entire storage system can lead to serious defects and unbalanced performance. For example, using only the 3-way Replication code could result in an unusually high storage overhead on the system [23], or using only the CRS code could result in a significant increase in read latency and, as a result, significantly higher repair times [3], [7]. Adaptive encoding LDSA-AE of Energy Internet can thus be achieved to join the merits of every code and meanwhile compensate their corresponding disadvantages, so as to achieve the optimal storage cost and system performance.

3 The Design of LDSA-AE

In this section, we first outline the main design objectives of LDSA-AE. Then we introduce its architecture overview, as well as the design and implementation of LDSA-AE. We also detail how to cluster Energy Internet data and provide solutions for distributed data distribution and recovery.

3.1 The Design Objectives of LDSA-AE

LDSA-AE is designed to achieve the following three goals.

- Reduce storage overhead. Replication strategy and MDR code are used by LDSA-AE to store the most accessed active data in the Energy Internet, aiming to reduce storage overhead while improving overall storage efficiency. In addition, although inactive data occupies a lower number of visits, it does not mean that they are not important, and using CRS code to store them aims to further reduce storage overhead.
- Increase repair rate. Once active data encounter damage and needs to be repaired, it can be quickly repaired through local or nearby replicas. When replicas are also corrupted or cannot be called in a timely manner, MDR code can be chunked to repair in parallel across servers, increasing repair speed while reducing access latency and thus avoiding the serious problem of failure to access. In addition, inactive data can be partitioned across servers for parallel repair by CRS code, which can effectively reduce access latency.
- Improves storage reliability. The Replication strategy and MDR code are used for data redundancy. Although the Replication strategy is not reliable, the MDR code can effectively compensate for the Replication strategy and ensure access to active data at any time, which further improves the fault tolerance of the system. Similarly, the storage reliability of the CRS code is fully adequate for the needs of inactive data.

Details about the EC storage system, RS code, CRS code, MDR code are shown in S2 in the Supplementary File of this paper.

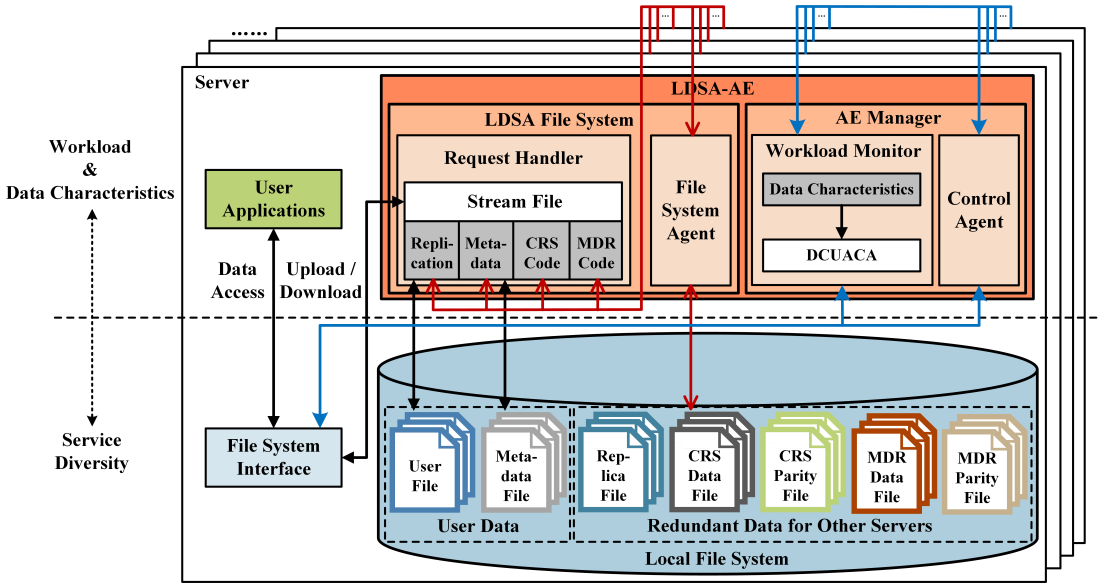

Fig. 2. LDSA-AE Architecture overview.

3.2 LDSA-AE Architecture Overview

LDSA-AE adopts *local file system-based redundant storage* (LFSRS) and adaptively changes the Replication strategy, CRS code, and MDR code according to the workload and data characteristics to achieve the effect of dynamic storage. LDSA-AE runs on each server and keeps in touch with LDSA-AE running on other servers all the time to realize redundant control among multiple servers. The overview of the LDSA-AE architecture is shown in Fig. 2.

First, LDSA-AE stores user data and file metadata in the local file system (FS), while the FS stores redundant data from other servers. In a system using LDSA-AE, the Replication strategy consists of r servers. LDSA-AE on each server duplicates the local user file and sends it to the corresponding $r-1$ other servers. When a parity file needs to be built, LDSA-AEs communicate with each other out of the $r-1$ other servers corresponding to the Replication strategy, with the goal of reducing the amount of redundant data and thus storage overhead. When the user data of the local FS is updated, LDSA-AE will also cooperate with the LDSA-AE of other servers to synchronously update the involved files.

Second, LDSA-AE itself is a freely stackable module on the local FS. LDSA-AE is a stackable module on top of a local FS. LDSA-AE consists of two main modules: LDSA FS, which handles file operations, and AE Manager, which performs adaptive encoding processing. LFSRS is in charge of LDSA FS, and AE Manager manages the switching of adaptive coding. Both modules will cooperate with LDSA-AEs running on other servers.

LDSA FS includes a request handler that handles user data access from the FS interface. It also includes an FS agent, which handles redundant data access from other servers. When there is a file access request, the FS interface sends the request to the request handler. Then, the request processor cooperates with the FS agents on other servers to update files on other servers in time, including replica files, block files for CRS and MDR codes, and parity files. In addition, LDSA FS also records all file metadata, which contains all configurations in the storage strategies, and we describe the details of distributed data distribution in Section 3.4.

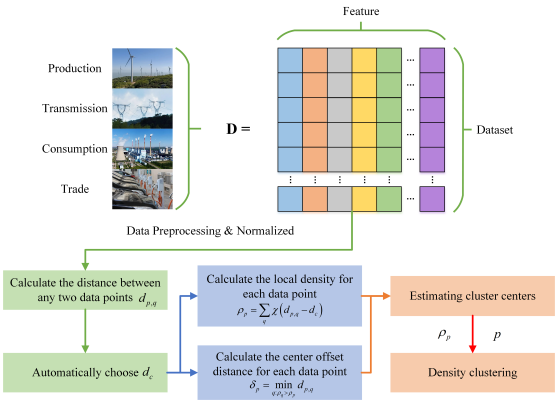

Fig. 3. Unsupervised automatic clustering algorithm.

AE Manager includes workload monitoring, which periodically collects monitoring data from LDSA FS. It also includes a control agent, which handles control requests from other servers. Workload monitoring has two main functional modules: data signature recognition and DCUACA. All user files are turned into data sets through the FS interface, and then the data sets processed by data feature identification are sent to DCUACA, and finally, the active data and inactive data are output by clustering. At this point, Workload Monitor is responsible for deciding which storage solution to use, maximizing the lightweight of the entire system. At the same time, AE Manager cooperates with other servers to perform adaptive encoding to ensure the reliability of data reconstruction in the event of server failure. We describe the details of DCUACA in detail in Section 3.3.

3.3 DCUACA

Since the Replication strategy and EC have different storage reliability, once the data clustering does not distinguish between active and inactive data in time, the resulting code-switching is not timely, which will lead to extremely unbalanced reliability of the storage system. DCUACA is

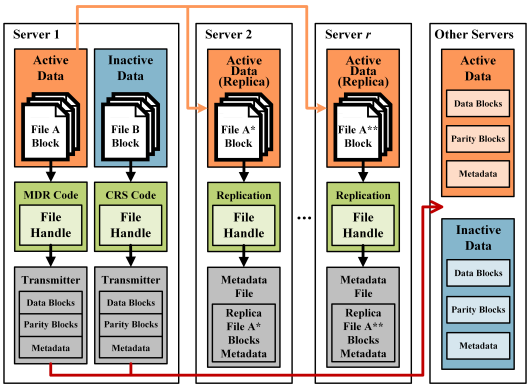

Fig. 4. LDSA-AE Distributed data distribution.

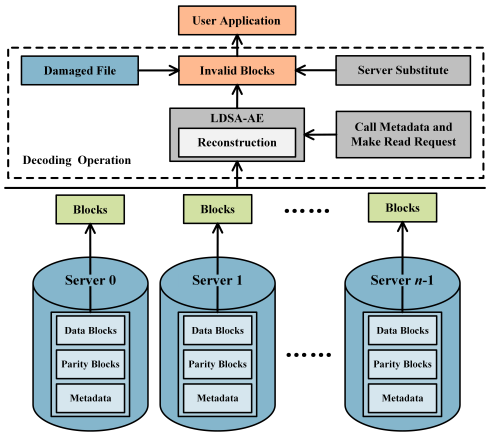

Fig. 5. LDSA-AE Efficient Reconstruction.

proposed in this paper to solve this problem. The algorithm is to detect non-spherical clusters and automatically find the correct number of clusters after detecting the distance between data points, and the local maximum of the density of data points is defined as the cluster center. In this process, there is no need to embed the data into a vector space in order to maximize the revealing of the density field at each data point. This paper assumes that adjacent points with lower local densities surround the cluster centers and they are relatively far away from any point with higher local densities. Fig. 3 shows the algorithm flow chart of DCUACA.

Definition 1. *Set the distance $d_{p,q}$ between each data point. Assuming that the triangle inequality is satisfied, d_c represents the cutoff distance, $\chi(\cdot)$ represents the cutoff distance function and satisfies $\chi(x) = \begin{cases} 1, & if\ x < 0 \\ 0, & otherwise \end{cases}$, then the local density ρ_p satisfies $\rho_p = \sum_q \chi(d_{p,q} - d_c)$, and the distance δ_p between the data point p and the higher density point satisfies $\delta_p = \min_{q:\rho_q > \rho_p} d_{p,q}$.*

ρ_p above is basically equal to the number of points that are closer to the data point p than d_c. The algorithm is only sensitive to the relative size of the different data points ρ_p, and it is obvious that the analysis results are robust in choosing d_c in larger datasets. However, $\delta_p = \max_q d_{p,q}$ is usually taken for the point with the highest local density, but this is only for the point with the local or global maximum value in the density. δ_p is still much larger than the typical nearest neighbor distance, so the point with an abnormally large δ_p value is identified as the cluster center. Meanwhile, the cutoff distance d_c is defined exogenously, but it can be chosen automatically with the help of rules of thumb in [24]. It can be seen that this is based on density connectivity implementation clustering and without the need to set the number of clusters in advance. It is suitable for clustering data sets with unknown content and large-scale, and can better shield the sensitivity to noise data.

3.4 Distributed Data Distribution

The distributed Replication strategy and recovery of failed data are intertwined and lack an efficient execution sequence. As a result, network bandwidth is occupied for a long time, making it difficult to optimize the storage cost of distributed storage. This paper optimizes the data distributed storage strategy of LDSA-AE. LDSA-AE stores user data and file metadata in the local FS. All servers manage five different files through LDSA-AE, including user files, replica files, data block files, parity block files, and metadata files. The distributed data distribution of LDSA-AE is shown in Fig. 4.

First, user files are plain native FS-based files that user applications can access directly, and DCUACA divides these files into active and inactive data. Previous studies [25] have shown that the Replication strategy is overwhelmingly advantageous if the system has a large amount of active data and requires frequent object access and minimal latency. However, the storage cost of Replication strategy is very high and the reliability cannot be guaranteed, so MDR code is needed to make up for the shortcomings of the system in these two aspects. On the other hand, for inactive data with low access frequency in the system, CRS code is used mainly considering the storage cost.

Second, LDSA-AE logically divides user files into fixed-size chunks to achieve high precision control between Replication strategy, MDR code, and CRS code. The total number of blocks of the MDR code and the CRS code is consistent. Algorithm 1 provides the workflow of LDSA-AE distributed data distribution. User files for active data are distributed directly to the other $r-1$ servers, and then replica metadata files are generated by the Replication file handler. At the same time, active data generates data block files, parity block files, and parity metadata through the MDR code file handler. It is then distributed through the transmitter, which communicates with each other beyond the $r-1$ other servers corresponding to the Replication strategy. Similarly, inactive data generates data block files, parity block files, and parity metadata files through CRS code file handlers. Both replica metadata files and parity metadata files contain reference pointers to each encoding block, and LDSA-AE can identify all blocks and parity files in the encoding through the metadata.

3.5 Efficient Reconstruction

Traditional erasure coding brings high-degraded read latency. Once reconstruction is required, the recovery time and recovery cost will increase exponentially, resulting in high-degraded repair throughput, which is time-consuming and economically inefficient. This paper proposes LDSA-AE efficient reconstruction for inpainting throughput. In the event of a server failure, LDSA-AE recovers lost data from

Algorithm 1: Distributed Data Distribution of LDSA-AE

Input: user files Q_s, r-way Replication, $[n, k_{mdr}]$ MDR code, $[n, k_{crs}]$ CRS code, servers S
Output: replicas Q^*, MDR code data blocks Q_{mdr}, MDR code parity blocks V_{mdr}, MDR code metadata MD_{mdr}, CRS code data blocks Q_{crs}, CRS code parity blocks V_{crs}, CRS code metadata MD_{crs}

Initialize $MD = 0$, $n = k_{mdr} + m_{mdr}$, $n = k_{crs} + m_{crs}$;
for *each* $i \in [0, s]$ **do**
  if Q_i *is active data* **then**
    store Q_i into local FS ;
    use r-way Replication to generate replicas Q_i^* ;
    for *each* $j = \{0, 1\}$ **do**
      AE Manager assigns Q_i^* to the corresponding S_j ;
    encode Q_i with MDR code to generate $Q_{mdr} = \left(q_{mdr_0}, q_{mdr_1}, \cdots, q_{mdr_{(k-1)}}\right), V_{mdr} = \left(v_{mdr_0}, v_{mdr_1}, \cdots, v_{mdr_{(m-1)}}\right), MD_{mdr} = \left(md_{mdr_0}, md_{mdr_1}, \cdots, md_{mdr_{(n-1)}}\right)$;
    for *each* $j \in [0, n-1]$ **do**
      AE Manager assigns $Q_{mdr}, V_{mdr}, MD_{mdr}$ to the corresponding S_j, which communicate with each other beyond the r other servers corresponding to r-way Replication ;
  else
    encode Q_i with CRS code to generate $Q_{crs} = \left(q_{crs_0}, q_{crs_1}, \cdots, q_{crs_{(k-1)}}\right), V_{crs} = \left(v_{crs_0}, v_{crs_1}, \cdots, v_{crs_{(m-1)}}\right), MD_{crs} = \left(md_{crs_0}, md_{crs_1}, \cdots, md_{crs_{(n-1)}}\right)$;
    for *each* $j \in [0, n-1]$ **do**
      AE Manager assigns $Q_{crs}, V_{crs}, MD_{crs}$ to the corresponding S_j ;
final ;
return $Q_i^*, Q_{mdr}, V_{mdr}, MD_{mdr}, Q_{crs}, V_{crs}, MD_{crs}$;

Algorithm 2: Efficient Reconstruction of LDSA-AE

Input: servers S, replica Q^*
Output: user file Q

Initialize AE Manager ;
if Q *is active data* **then**
  if Q^* *is valid* **then**
    replicate Q^* to the server substitute ;
  else
    AE Manager calls metadata of Q and make read request ;
    find z failed servers in S ;
    send Q generated by decoding with MDR code to server substitute ;
    once the failed servers S_z recover ;
    for *each* $i \in [0, z-1]$ **do**
      place the recovered data block q_{mdr_z} or patity block v_{mdr_z} on the corresponding S_z ;
else
  AE Manager calls metadata of Q and make read request ;
  find z failed servers in S ;
  send Q generated by decoding with CRS code to server substitute ;
  once the failed servers S_z recover ;
  for *each* $i \in [0, z-1]$ **do**
    place the recovered data block q_{mdr_z} or patity block v_{mdr_z} on the corresponding S_z ;
final ;
return Q;

other remaining data blocks and parity blocks. Algorithm 2 provides a workflow for the efficient reconstruction of LDSA-AE. For replica files of active data, you can directly call metadata files to view the corresponding replica files, and then replicate replica files to the server substitute to reconstruct the lost data through replica files.

If replica files are invalid during active data reconstruction, or the cost of replicating replica files is too high, the data file generated by MDR code must be enabled for timely reconstruction. First, AE Manager will call metadata files to view all the servers of n storage blocks and issue a read request to retrieve the data in the storage system. If any of the storage server's blocks become invalid, a decode operation is performed to reconstruct the data file from these remaining blocks. In the event of a server failure, data redundancy is temporarily reduced until the failed server recovers. In addition, if the lost contains parity blocks, the lost parity blocks also need to be reconstructed to restore the redundancy of the user data. Fortunately, LDSA-AE can quickly recover redundant storage of data in the same way as generating replicas, once lost data blocks and parity blocks have been reconstructed. At the same time, the inactive data reconstruction process is equivalent to that of active data when replica files are invalid. The efficient reconstruction of LDSA-AE is shown in Fig. 5.

4 Theoretical Analysis

At present, the LDSA-AE we designed mainly addresses the issue of storage strategy and recovery technology and optimizes the storage overhead at the same time. However, we believe that recovery performance is the more important factor since the recovery I/O of a $[n, k]$ EC system requires k times that of a Replication strategy.

4.1 Fixed-size Chunks

Since the data generated by the Energy Internet can be as much as tens of petabytes of data, which poses a huge challenge to storage systems. So we expect a large number of failures, so high fault tolerance and high reliability are

critical. In order to provide high data reliability, LDSA-AE of each server will monitor the health status of each storage server 24 hours a day and find faults in time.

In a storage system, data is partitioned and stored as a number of fixed-size blocks, which are the smallest unit of data that constitutes access.

At the same time, the storage system needs to write the maximum input data size of each task to the composite workload. The input data size has the same granularity as the file size, which we set to 64MB for LDSA-AE, the same as the default HDFS block size [26]. We think this setup is reasonable because our input files will be as granular as the underlying HDFS, and the overhead of reading concurrent jobs from the same HDFS input is negligible [27].

4.2 Choice of Replication Strategy

In the case of replicating the entire user file, the overhead is 100% for a single replica, 150% for two replicas, and so on. It can be seen that the storage overhead of the Replication strategy is huge, even if it can bring the fastest repair rate. Obviously, the higher the degree of Replication, the better. But the higher the degree of Replication, the higher the cost of storage overhead and write/update latency. Therefore, there is a trade-off between cost and performance, and currently, in our LDSA-AE design, it is wise to choose 3-way Replication for the Replication strategy. It is also explicitly demonstrated in [28] that for systems requiring very high reliability, 3-way Replication can actually guarantee that data will never be lost.

4.3 Storage Cost

CRS code encoding is a process that protects data by storing individual user files in slices. Therefore, data protection can be realized with higher storage efficiency. It can be seen that the storage cost occupied by CRS code storage is less than that of a single replica, that is, the cost is less than 100%. MDR code is applied to RAID-6 scenarios, and it is necessary to calculate the content of the encoded bit, look at the row of the bit in the system encoding matrix, and calculate the XOR of each data bit. It can be seen that the storage overhead occupied by MDR code is larger than that of CRS code, but it is still lower than Replication strategy.

4.4 Data Reliability

Mean time between failures (MTBF) is a reliability quantification index in "hours". It reflects the time quality of a disk, and the ratio of the accumulated working time of a disk to the number of failures in the total use phase is MTBF. MTBF is usually used for a fault-recoverable system and refers to the average time between two failures of the system, also known as the mean interval between failures. Simply put, fewer disk failures, that is, higher MTBF means higher disk reliability.

Definition 2. *Let $P(t,m)$ represent the probability of consecutively damaging m disks in time t, obeying the Poisson distribution, then there is $P(t,m) = \frac{(\lambda t)^m \cdot e^{-\lambda t}}{m!}$. We call $\mathrm{MTBF}(t,m) = \frac{365\cdot 24}{1-(1-P(t,m))^{365\cdot 24/t}}$ the MTBF of the data recovery system.*

On the basis of Definition 2, for different storage systems, we obtain the following mean time between failures.

4.4.1 Replication strategy

Suppose that the Replication strategy storage system contains r replicas, the annual disk failure probability is P, the slice size is T, the number of disks is N, the capacity of a single disk is $perD$, and the recovery time of a single disk failure is t. The entire cluster has $C(N,r) = N!/(r!\cdot(N-r)!)$ combinations of r replicas. According to Definition 2, we have the MTBF of r replicas as

$$\mathrm{MTBF}_r = \frac{365\cdot 24}{1-\left(1-\frac{N\cdot perD}{T}\cdot\frac{1}{m}\cdot\frac{1}{C_N^m}\cdot P(t,m)\right)^{365\cdot 24/t}}. \quad (1)$$

4.4.2 CRS code

Suppose the CRS code storage system contains n disks, of which k are data disks, P is the annual disk failure probability, T is the slice size, N is the number of disks, $perD$ is the usable capacity of a single disk, and t is the recovery time for a single disk failure. By allowing at most $m-1$ blocks lost, there are combinations of $C(N,n) = N!/(n!\cdot(N-n)!)$ EC groups in the entire cluster. According to definition 2, we get the MTBF of the $[n,k]$ CRS code as

$$\mathrm{MTBF}_{crs} = \frac{365\cdot 24}{1-\left(1-\frac{N\cdot perD}{T}\cdot\frac{k+1}{n}\cdot\frac{C_n^m}{C_N^m}\cdot P(t,m)\right)^{365\cdot 24/t}}. \quad (2)$$

4.4.3 MDR code

Suppose the MDR code storage system contains n disks, of which k are data disks, the annual disk failure probability is P, the number of disk groups is N, the recovery time for a single disk failure is t, and the number of disks in a single level is n. Then according to the Markov reliability model of RAID [29], the system of simultaneous differential equations can be obtained

$$\mathrm{MTBF}_{mdr} = \left(\frac{365\cdot 24}{P(t,m)}\right)^3 / \left[N\cdot n\cdot(n-1)\cdot(n-2)\cdot t^2\right]. \quad (3)$$

Given a value of n, we observe that the reliability of a single disk is 1) improved with an increasing m and a constant k and 2) imperiled with an increasing k and a constant m. While in general, increasing k results in a higher order of magnitude reliability improvement than increasing m.

4.5 Read Latency

If all replicas and all encoding blocks are located on the same server, the latency problem is not as severe as when replicas and encoding blocks are geographically separated.

In Replication strategy, a request is made to read the nearest available replica. If the most recent replica is unavailable, a request is made to read the next most recent replica, and so on, until either a replica becomes available or the request fails. In $[n,k]$ CRS code and MDR code, the request can be answered by reconstructing the user file from the k nearest data blocks. As requests encounter recoverable disk failures more often in EC, there is usually an additional read latency.

Lemma 1. *Given a Replication strategy storage system, r user file replicas and the probability that each replica is unavailable is p_r,*

and the latency of the server requesting data is L. When $p_r \to 0^+$, we have

$$\underset{p_r \to 0^+}{RL_r} = (1 - p_r) \cdot L_1 + p_r \cdot L_2. \tag{4}$$

Proof. Assuming each replica is on a different server, the latency for requesting data from these servers is $L_1 < L_2 < \cdots < L_r$ for a given user client. Then with probability $1 - p_r$, the latency to request data will be L_1. With probability $p_r \cdot (1 - p_r)$, the delay will be L_2. We can get the expected delay will be $RL_r = \sum_{i=1}^{r} {p_r}^{i-1} \cdot (1 - p_r) \cdot L_i$. If p_r is fairly small, the high-power terms of p_r can be ignored, and only the first two terms are kept, which can be simplified to $RL_r = (1 - p_r) \cdot L_1 + p_r \cdot L_2$.

The proof is completed. □

Example 1. *Suppose a user file is unavailable on the nearest server with a probability of 0.002, and the second server has a latency 100 times greater than the nearest server. By $RL_r = (1 - p_r)\, L_1 + p_r \cdot (L_1 \cdot 100) = .998 + .2 = 1.198$, the expected latency of the second server increases by about 20%.*

Lemma 2. *A given EC storage system contains n disks, of whichk are data disks. The probability that each disk is unavailable is p_{ec}, and the latency of the server requesting data is L. When $p_{ec} \to 0^+$, we have*

$$\underset{p_{ec} \to 0^+}{RL_{ec}} = (1 - p_{ec}) \cdot L_1 + k \cdot p_{ec} \cdot L_2. \tag{5}$$

Proof. Assume that each data disk is located on a different server. If k blocks are stored in one location, the probability of one local disk failure is $k \cdot p$. This means that the probability of a single local failure of the EC and the necessity to request a server that is farther away is k times greater than the Replication strategy, and we can get that the expected latency will be $RL_{ec} = \sum_{i=1}^{k} (k \cdot p_{ec})^{i-1} \cdot (1 - p_{ec}) \cdot L_i$. If p_r is fairly small, the higher power terms of p_r can be ignored, and only the first two terms are kept, which can be simplified to $RL_{ec} = (1 - p_{ec}) \cdot L_1 + k \cdot p_{ec} \cdot L_2$.

The proof is completed. □

Example 2. *Suppose the nearest server has a latency of $L_1 = 1$. If k=6, the expected latency using the Replication strategy is 1.198, and the expected latency using EC is $RL_{ec} = (1 - p_{ec}) \cdot L_1 + k \cdot p_{ec} \cdot L_2 = 0.998 + 1.2 = 2.198$.*

For active data and frequently accessed user files, latency is a major concern and the latency advantage of Replication strategies should be considered. For inactive data, where user files are archived and rarely accessed, latency may not be a major consideration.

5 Performance Evaluation

In this section, we conduct simulation experiments based on the HDFS platform from three aspects of storage reliability, storage overhead, and repair throughput to verify the performance of the proposed algorithm. **The experimental environment is shown in S3 in the Supplementary File of this paper.** The four servers are all connected through a 1 Gb/s network, which is the common bandwidth of the network edge [30], and each machine runs up to 16 client sides.

Benchmark: To measure performance, we use the Hammer-Bench [31] benchmark test tool to generate file access, a distributed application that can create up to tens of thousands of HDFS client sides and spread across multiple servers. With Hammer-Bench, you can test the performance of file system workloads based on industrial workload tracing. **The production tracing survey in a synthetic workload based on 1-hour samples [32] is shown in S5 in the Supplementary File of this paper.** We then evaluate synthetic workloads for active and inactive data, both sampled from the production tracing survey. **The detailed description of workloads is shown in S6 in the Supplementary File of this paper.** We find that active data is in an absolute position. Overall, it accounts for more than 93.8% of the performance overhead, while inactive data consumes less overhead. This means that the scope and frequency of energy big data acquisition have increased significantly, and the proportion of unstructured data such as audio and video has further increased [33].

Second, in our next experiments, we aim to answer the following research questions (RQs).

- RQ1: Will changes in data buffer size affect encoding performance?
- RQ2: Is there a better cache effect with smaller packets?
- RQ3: Is the distinction between active data and inactive data obvious?
- RQ4: Does LDSA-AE significantly improve the reliability of data storage in the Energy Internet?
- RQ5: Can LDSA-AE effectively reduce the data storage overhead of the Energy Internet?
- RQ6: Can LDSA-AE effectively improve the restoration throughput of data storage in the Internet of Energy?

We describe the encoding and decoding performance comparison tests of RS, CRS, and MDR codes in Section 5.1, including RAID-6 and non-RAID-6 scenarios. In all experiments, they are coded as $n = k + m$ combinations, and the storage system can tolerate up to m disk failures. The encoding experiment requires the encoder to read the user file, encode it and write it t $k + m$ encoded blocks, thus measuring the performance of the encoding operation. For the decoding experiments, we measure the performance of the decoding operation against only missing data blocks, decoding the data driver and testing the recovery encoding rate.

Both encoding and decoding experiments use [n,k] to represent the encoding block combination, and we set four combinations of n and k. Two of these combinations are RAID-6 schemes: [8,6] and [16,14]. The other two combinations represent the more fault-tolerant 16-disk stripes: [16,12] and [16,10]. These combinations are relatively common in practical use, and although large storage systems consist of a large number of disks, the stripe size tends to stay around this range. Because large stripes bring diminishing returns and higher memory footprint compared to storage overhead, both [34] and [35] demonstrate that keeping to 16 and below is optimal, so we stripe Size stays at 16 or less.

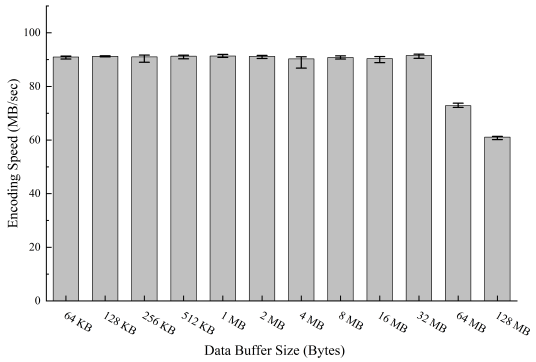

Fig. 6. Encoding speeds to read a 256 MB file, where $k = 10$, $m = 6$, $w = 8$.

The actual effect of DCUACA is tested in Section 5.2 with a dataset containing energy-related structured, semi-structured, and unstructured data. The metadata of the dataset content includes access ID (numeric key), access name (string), access time (slot seconds), upload/download file size (bytes), submission time (slot seconds), and upload/download file path (string). Here each digital feature is the working dimension, but some features have relatively weak or even unusable effects on the results, so normalization is required after data preprocessing.

Finally, the three challenges proposed in Section I are evaluated and demonstrated in Section 5.3. Here we have added several sets of performance comparison experiments, including the performance comparison of LDSA-AE and traditional single-method encoding and combined encoding.

5.1 Confirmatory Experiment

5.1.1 *Impact of Buffer Size (RQ1)*

The purpose of testing large file encoding is to verify the actual performance of encoding large files, since performing a large amount of I/O can cause a significant change in performance. We use a 256 MB video file as a test, RS code $k = 10$, $m = 6$, $w = 8$ (because 1bit = 8 bit, this is the best computer speed), packet size = 1024 Bytes, buffer size from 64 KB to 256 MB, up to the file size of the video itself. Fig. 6 shows the test results for the buffer.

Each buffer in Fig. 6 is the result of ten executions, the average value is shown, and the error range is also marked in the figure. It can be seen from Fig. 6 that the impact of modifying the buffer size on our subsequent experimental tests can be ignored. Therefore, in the following experimental tests, we kept the data buffer size of about 128 KB. In fact, the size of the buffer depends on many factors, the processor must be multi-tasking and multi-threading, and cannot be fixed to a constant value. We fixed the buffer size in the range of 128 KB, which is enough to support efficient I/O operations without consuming all the memory of the computer.

5.1.2 *Impact of Packet Size (RQ2)*

For better cache performance, we use smaller packet sizes because we could get a less tight XOR loop that would otherwise cause more buffer failures. We use a 256 MB video file for testing, CRS code is $k = 6$, $m = 2$, $w = 8$, and the packet size ranges from 4 to 10000 Bytes. In this test, one set of data is taken after repeated operation 10 times, and the repetition rate of each set of data is guaranteed to be within 0.1%. Fig. 7 shows the test results for packet size.

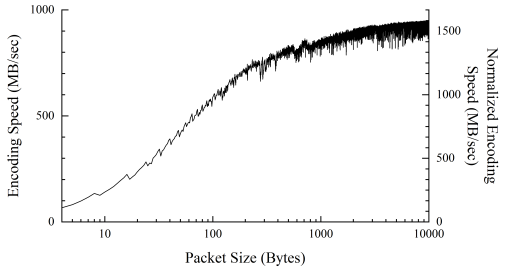

Fig. 7. The effect of modifying the packet size on CRS coding, where $k = 6$, $m = 2$, $w = 8$.

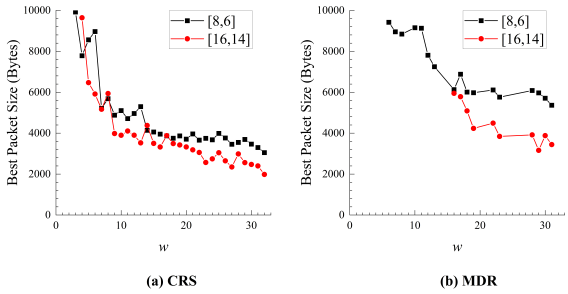

Fig. 8. The effect of modifying w on the best packet sizes found.

There are two y-axes in Fig. 7. On the left is the encoding speed, which is the size of the input file divided by the time it takes to encode; on the right, we normalize the encoding speed so that we can compare the encoding performance of different algorithms.

Definition 3. *Assuming that the number of data blocks is k, the number of parity blocks is m, and $encodingspeed$ is the speed of encoding the input file, we can obtain the normalized encoding speed and calculate it as follows:*

$$\begin{matrix} Normalized \\ Encoding\ Speed \end{matrix} = \frac{Encoding\ Speed \cdot m \cdot (k-1)}{k}. \quad (6)$$

In general, higher packet sizes have better performance. However, the highest performance is achieved when the code used makes full use of buffering. In the packet size test, the optimal packet size of [8,6] CRS code combination is 9008 Bytes, which can achieve an encoding speed of 949.657 MB/sec, that is, a normalized encoding speed of 1582.762 MB/sec. In fact, this curve does not monotonically increase to its optimum value, nor does it monotonically decrease. To make matters worse, there can be sharp declines in performance between adjacent packet sizes due to collisions between buffer entries. For example, at packet sizes 5716, 5717, and 5718 Bytes, the normalized encoding speed are 1549.693, 1388.885, and 1558.003 MB/sec respectively.

The optimal packet size decreases as k, m, and w increase, as these variables all increase the stripe size. Therefore, smaller packets are required to fit most of the stripes into the buffer. We explore the optimal packet sizes for different w in Fig. 8, where CRS codes and MDR codes are tested. Since

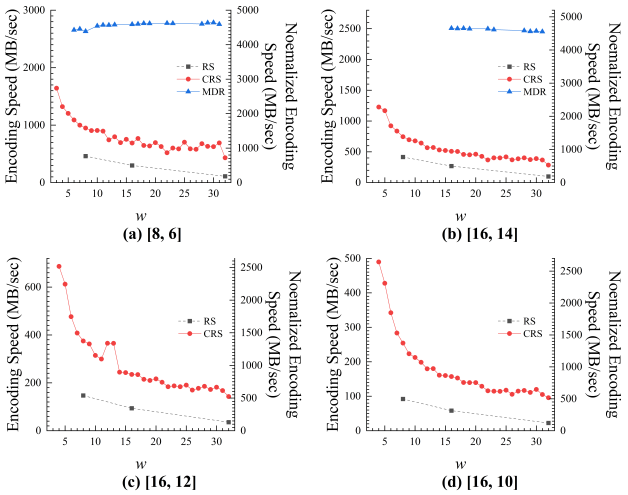

Fig. 9. Encoding performance.

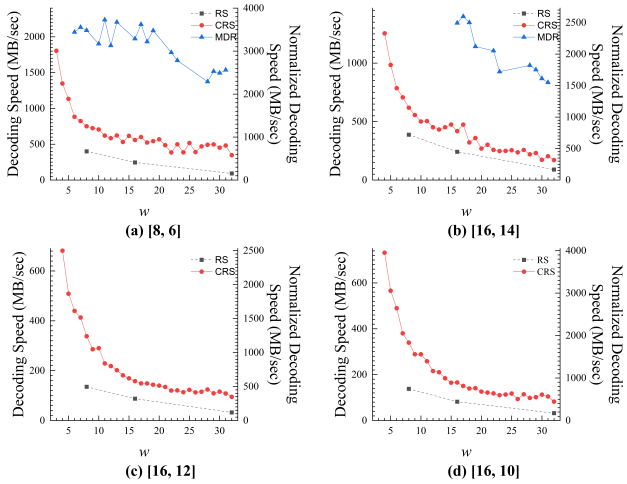

Fig. 10. Decoding performance.

$m = 2$ is optimal in the RAID-6 scenario, the impact of m changes will not be discussed in this test session. For each code, the larger the value of k, the smaller the packet size, and the optimal packet size tends to decrease as w increases.

5.1.3 *Overall Encoding Performance*

Now, we present the performance of each code encoding. In the selection of the packet size of the code, we choose from the experimental results of the best packet size test. The results for the [8,6] configuration are shown in Fig. 9a, and the results for the [16,14] configuration are shown in Fig. 9b. The graphs for both configurations look similar, and they both share some characteristics. MDR codes are better than all other codes because it is in the RAID-6 scenario, and the performance of the optimal code is independent of k, which means it has good flexibility. Second, a larger w value performs better than a smaller w value, but the impact of w changes can be ignored. This result is expected because smaller values of w require the encoding engine to store fewer blocks of data in memory, and larger w performs better due to memory and caching effects.

Although the choice of w is very important, the performance of CRS codes is also excellent. However, when $w \leq 8$, the encoding speed of CRS codes drops significantly. The choice of $w \in \{8, 16, 32\}$ are quite natural choices since they allow strip sizes to be powers of two. However, since the number of ones in the CRS generator matrix depends on the number of bits in the original polynomial of the Galois Field, the polynomial of $w \in \{8, 16, 32\}$ has one more bit than the other polynomials, resulting in worse performance.

Fig. 9c and Fig. 9d show the results for the [16,12] and [16,10] configurations, respectively. Since this is no longer a RAID-6 scenario, only RS and CRS codes are shown. The normalization performance of CRS encoding is much worse now because the generator matrix is denser and cannot be optimized like when $m = 2$. The definition of normalized encoding speed means that its normalized encoding speed should match the XOR speed to be called optimally encoded. Therefore, compared to the [16,14] configuration, the codes in the [16,12] and [16,10] configurations need to perform more XOR operations, which is why the normalized encoding speed is much slower than in the RAID-6 scenario.

5.1.4 *Decoding Experiments*

To test decoding performance, this test will decode the encoder program. Specifically, the decoder selects m random data drives, and after each encoding iteration, it zeros the buffers of these drives before decoding. This test only decodes data drives for two reasons. First, it represents the most difficult case to decode, because once a block of data is lost, all encoding information must be recalled; second, most decoding programs only decode data and re-encode a single code strip if all code strips are re-encoded. While it is possible to modify these programs for the purpose of individual re-encoding, this is not in the spirit of the test evaluation. Therefore, the test code is written to check whether the erased data is decoded correctly before the test.

Fig. 10a and Fig. 10b demonstrate the performance of the [8,6] and [16,14] configurations, respectively. MDR code is slightly more complicated to decode, but its efficient decoding performance greatly benefits from Code-Specific Hybrid Reconstruction [36]. Without optimization, its decoding performance will be extremely inefficient. CRS code also enjoys the benefits of optimization. As for the performance of the RS decoder, it is the same as the other corresponding encoders, since the only difference between RS encoding and decoding is the $k * k$ matrix inversion.

Fig. 10c and Fig. 10d show the performance of the [16,12] and [16,10] configurations, respectively. It can be seen that in the non-RAID-6 scenario, the decoding performance trend of CRS code is the same, although as w becomes larger, the decoding performance becomes very low. At the same time, the performance of RS encoding remains the same as decoding.

5.2 DCUACA Experiments (RQ3)

The performance evaluation of the DCUACA algorithm is shown in S4 in the Supplementary File of this paper. Fig. 11 shows the distribution of the access frequency of Energy Internet data obtained by DCUACA, and the abscissa

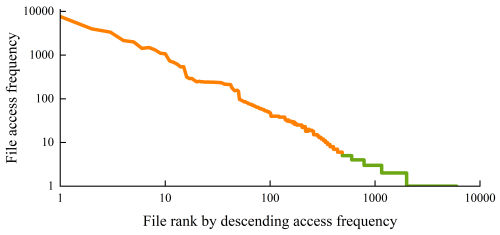

Fig. 11. Dataset file access frequency and level.

is ranked according to the non-decreasing frequency. Note that the distribution is plotted on a logarithmic axis, rather than forming an approximately monotonically decreasing line. This reveals the fact that the data access frequency of the Energy Internet follows a Zipf distribution and most data access flows to a small subset of data. Most data is accessed so infrequently that active data takes up most of the data access and inactive data takes up most of the storage space.

Next, from Fig. 11 we can see that there are two clusters, which is due to the fact that we specify the number of labels for DCUACA to be 2. The distinction between the two clusters is obvious. Data files lower than file level 487 have higher access frequency, and this part is active data. On the contrary, the access frequency of the lower part is relatively low. There are even close to 30% of file grades behind, and their access frequency is only 1.

5.3 LDSA-AE Comparative Experiments

LDSA-AE uses a reliable data storage scheme based on CRS code for the storage of inactive data of the Energy Internet, and a reliable data storage scheme combining Replication strategy and MDR code for the storage of active data of the Energy Internet. Here we add several sets of coding performance comparison experiments, including traditional single-method coding: 3-way Replication strategy, CRS code, and MDR code; combined coding: combined Replication strategy and RS code (RRS) [37].

5.3.1 *Data Storage Reliability Performance (RQ4)*

Firstly, we analyze the data reliability of the system. For simplicity, we assume that failures are only on disks, with no other failure correlations. Assuming that the capacity of a single disk is 6 T, the utilization rate is 70%, the data slice size is 1 GB, the failure recovery time of a single disk for 3-way Replication is 1 hour, RS code and CRS code is 3 hours, and MDR code is 5 hours. In a cluster with a total of 64 disks, the MTBF of different codes is tested and compared. Fig. 12a shows the MTBF test results. It can be seen from Fig. 12a that the MDR code maintains the highest level of MTBF, followed by LDSA-AE, which is several orders of magnitude more reliable than the remaining three codes. RRS code and CRS code show a more pronounced downward trend, even lower than 3-way Replication when $n \geq 34$, which happens because the repair performance of RS and CRS codes decrease rapidly as k increases. In other words, for disks with a fixed level of reliability and redundancy, EC takes up less disk space than a Replication strategy, or stores more data on the same disk space and provides the same data reliability protection. The reliability of 3-way Replication is not affected by these parameters. With the increase of k, the storage reliability of RS codes and CRS codes was once lower than that of the Replication strategy. In short, the data storage reliability of LDSA-AE is as expected and is of a very high standard.

5.3.2 *Storage Overhead Performance (RQ5)*

Now we observe how the storage overhead changes as $[n, k]$ grow. We compared 5 codes: 3-way Replication, CRS code, MDR code, LDSA-AE, and RRS. To keep the storage overhead easy to understand, we define the cost of storing a byte as a measure of how many bytes are stored per useful byte. Obviously, high cost leads to high storage overhead. Since 3-way Replication is a commonly used method, the critical Replication strategy due to its own properties, extremely high storage overhead is unavoidable, so we use it as a baseline for comparison. The results are shown in Fig. 12b, when $[n, k]$ is fixed, the normalized cost of all codes except 3-way Replication decreases as n increases. CRS code maintains the lowest storage overhead, followed by LDSA-AE, MDR, and RRS, but MDR code is second only to 3-way Replication when $n = 10$. Compared with RRS, the improvement in storage overhead of LDSA-AE is obvious. On the whole, LDSA-AE takes advantage of the local storage advantages of LFSRS to the greatest extent, which is the key to ensuring that it always maintains the storage overhead second only to the CRS code.

5.3.3 *Repair Throughput Performance (RQ6)*

In this test session, we measured the throughput of repairing a failed data server. A total of 4 machines are involved in the experiment. Each machine stores about 100 GB of data. The data is partitioned and stored in fixed-size blocks. We chose 64 MB or about 1600 encoded blocks. We fail a random machine and start the data repair process. After the repair is complete, measure the elapsed time and calculate the repair throughput. The results are shown in Fig. 12c. As can be seen from Fig. 12c, the repair performance of 3-way Replication is the best, and it is worth noting that the throughput of 3-way Replication is constant over different $[n, k]$. It is followed by LDSA-AE, RRS, MDR code, and CRS code has the worst performance. This is not surprising since MDR code and CRS code have to access a lot of data to be repaired. But the performance of the MDR code gets worse as n increases and is even lower than the CRS code when $n = 52$. In addition, the repair throughput of MDR code is second only to 3-way Replication when $n = 10$. Overall, the repair throughput of LDSA-AE is still very good, which is mainly due to the flexible scheduling control of AE Manager. The existence of AE Manager allows applications to directly access data copies in local storage, which is crucial for active data access.

6 Conclusions

This paper proposed a novel adaptive dynamic encoding algorithm, termed LDSA-AE, to alleviate the degraded read latency in distributed storage systems, thereby championing an efficient and secure data storage apparatus for Energy Internet. Our key idea is to leverage DCUACA to realize a real-time classification of active and inactive data without supervision, balancing the tradeoff between storage and

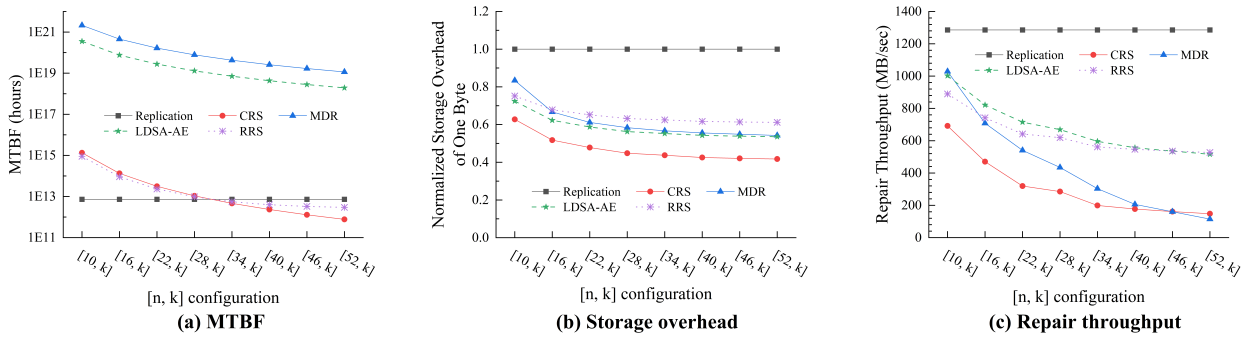

Fig. 12. LDSA-AE Comparisons, where $m = 4$ of CRS code, $m = 2$ of MDR code, $m = 4$ of RS code.

communication overheads. Specifically, for active data, the Replication strategy can call the replicas with the fastest efficiency to achieve recovery performance, assisted by the MDR codes, which achieve reliable storage through the auxiliary Replication strategy, thereby reducing the read degradation delay without sacrificing reliability. For inactive data, CRS codes have the advantage of low storage overhead without sacrificing more reliability and latency. In addition, we exploit the adaptive encoding to ensure high redundancy efficiency at the same performance level and relieve storage pressure. Control code parameters are employed to avoid unacceptable performance degradation. Experiments benchmarked on real testbed substantiated the superiority of LDSA-AE in terms of reliability, storage overhead, and repair throughput. Also, by relaxing storage constraints at a reasonable scale, LDSA-AE can ensure remarkable fault tolerance and efficient repair with low communication latency. We envision this study can shed some light on the data security research for smart energy systems. As future work, we shall study the generalizability of our LDSA-AE encoding algorithm for other cyber-physical systems with heterogeneous data modalities and long-tailed user access distributions, and evaluation of the proposed scheme under other benchmarks like YCSB [38] is the future work.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under Grant No. 51977113, 62293500, 62293501, 62293505, and 62176070. Postgraduate Research & Practice Innovation Program of Jiangsu Province under Grant No. SJCX21_0291, and BAGUI Scholar Program of Guangxi Zhuang Autonomous Region of China under Grant No. 201979.

REFERENCES

[1] K. Zhou, S. Yang, and Z. Shao, "Energy internet: the business perspective," *Applied Energy*, vol. 178, pp. 212–222, 2016.
[2] T. A. S. Team, "Amazon s3 availability event," Available:https://status.aws.amazon.com/s3-20080720.html, Jul. 20, 2008.
[3] R. K. Ko, M. Kirchberg, and B. S. Lee, "From system-centric to data-centric logging-accountability, trust & security in cloud computing," in *2011 Defense Science Research Conference and Expo (DSR)*. IEEE, 2011, pp. 1–4.
[4] K. Zetter, "Everything we know about ukraine's power plant hack," *Wired*, 2016.
[5] T. Miller, A. Staves, S. Maesschalck, M. Sturdee, and B. Green, "Looking back to look forward: Lessons learnt from cyber-attacks on industrial control systems," *International Journal of Critical Infrastructure Protection*, vol. 35, p. 100464, 2021.
[6] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–12.
[7] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *2012 {USENIX} Annual Technical Conference*, 2012, pp. 15–26.
[8] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster," in *5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 13)*, 2013.
[9] A. O. S. Service, http://www.aliyun.com/.
[10] A. Wang, "Apache hadoop 3.0.0. apache software foundation," RetrievedMarch3,2020fromhttps://hadoop.apache.org/docs/r3.0.0/, 2017.
[11] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 307–320.
[12] H.-Y. Lin and W.-G. Tzeng, "A secure decentralized erasure code for distributed networked storage," *IEEE transactions on Parallel and Distributed Systems*, vol. 21, no. 11, pp. 1586–1594, 2010.
[13] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *arXiv preprint arXiv:1208.4174*, 2012.
[14] Z. Wang, H. Wang, A. Shao, and D. Wang, "An adaptive erasure-coded storage scheme with an efficient code-switching algorithm," in *49th International Conference on Parallel Processing-ICPP*, 2020, pp. 1–11.
[15] J. Li, J. Nelson, E. Michael, X. Jin, and D. R. Ports, "Pegasus: Tolerating skewed workloads in distributed storage with {In-Network} coherence directories," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 387–406.
[16] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.
[17] Z. Wang, T. Li, H. Wang, A. Shao, Y. Bai, S. Cai, Z. Xu, and D. Wang, "Craft: An erasure-coding-supported version of raft for reducing storage cost and network cost," in *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*, 2020, pp. 297–308.
[18] L. Xu, M. Lyu, Z. Li, Y. Li, and Y. Xu, "Deterministic data distribution for efficient recovery in erasure-coded storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2248–2262, 2020.
[19] Z. Yuan, X. You, X. Lv, M. Li, and P. Xie, "Hds: optimizing data migration and parity update to realize raid-6 scaling for hdp," *Cluster Computing*, vol. 24, no. 4, pp. 3815–3835, 2021.
[20] T. Zhou and C. Tian, "Fast erasure coding for data storage: A comprehensive study of the acceleration techniques," *ACM Transactions on Storage (TOS)*, vol. 16, no. 1, pp. 1–24, 2020.
[21] R. Nachiappan, B. Javadi, R. N. Calheiros, and K. M. Matawie, "Adaptive bandwidth-efficient recovery techniques in erasure-

coded cloud storage," in *European Conference on Parallel Processing*. Springer, 2018, pp. 325–338.
[22] H. Bao, Y. Wang, and F. Xu, "Reducing network cost of data repair in erasure-coded cross-datacenter storage," *Future Generation Computer Systems*, vol. 102, pp. 494–506, 2020.
[23] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Transactions on Database Systems (TODS)*, vol. 22, no. 2, pp. 255–314, 1997.
[24] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *science*, vol. 344, no. 6191, pp. 1492–1496, 2014.
[25] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, "Simple regenerating codes: Network coding for cloud storage," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 2801–2805.
[26] D. Borthakur *et al.*, "Hdfs architecture guide," *Hadoop apache project*, vol. 53, no. 1-13, p. 2, 2008.
[27] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *2011 IEEE 19th annual international symposium on modelling, analysis, and simulation of computer and telecommunication systems*. IEEE, 2011, pp. 390–399.
[28] Q. Xin, E. L. Miller, T. Schwarz, D. D. Long, S. A. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," in *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings*. IEEE, 2003, pp. 146–156.
[29] J.-j. Lu, X. Zhang, and X.-P. Zhou, "Based on markov model analyze a new raid reliability," in *2012 7th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2012, pp. 211–213.
[30] R. Li, X. Li, P. P. Lee, and Q. Huang, "Repair pipelining for {Erasure-Coded} storage," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 567–579.
[31] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt, and M. Ronström, "{HopsFS}: Scaling hierarchical file system metadata using {NewSQL} databases," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017, pp. 89–104.
[32] K. Li, Y. Tang, J. Chen, Z. Yuan, C. Xu, and J. Xu, "Cost-effective data feeds to blockchains via workload-adaptive data replication," in *Proceedings of the 21st international middleware conference*, 2020, pp. 371–385.
[33] K. Wang, Y. Wang, X. Hu, Y. Sun, D.-J. Deng, A. Vinel, and Y. Zhang, "Wireless big data computing in smart grid," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 58–64, 2017.
[34] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage." in *Fast*, vol. 8, 2008, pp. 1–16.
[35] B. Welch, M. Unangst, Z. Abbasi, G. A. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system." in *FAST*, vol. 8, 2008, pp. 1–17.
[36] J. L. Hafner, V. Deenadhayalan, K. Rao, and J. A. Tomlin, "Matrix methods for lost data reconstruction in erasure codes." in *FAST*, vol. 5, no. December, 2005, pp. 15–30.
[37] B. A. Ignacio, C. Wu, and J. Li, "Warmcache: A comprehensive distributed storage system combining replication, erasure codes and buffer cache," in *International Conference on Green, Pervasive, and Cloud Computing*. Springer, 2018, pp. 269–283.
[38] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

Song Deng (M'16) received the Ph.D.degree in information network from Nanjing University of Posts and Telecommunication, Nanjing, China, in 2009. From 2009 to 2012, he was a Research Fellow with the State Grid Electric Power Research Institute, Nanjing, China. From 2012 to 2014, he was a Research Fellow with the China Electric Power Research Institute, Beijing, China. He is currently the Associate Professor of Nanjing University of Posts and Telecommunication, Nanjing, China. He was an international visitor with computer science from the University of Louisiana at Lafayette, USA, from 2018 to 2019. His research interests include distributed computing, data reliable storage for CPS, data mining and knowledge engineering.

Yujia Zhai received the B.E. degree in automation from Chengxian College of Southeast University in Nanjing, China in 2020. He is currently pursuing the M.S. degree in computer science at Nanjing University of Posts and Telecommunications, Nanjing, China. His research interests include power grid data security and data reliability recovery.

Di Wu (M'19) received his Ph.D. degree from the Chongqing Institute of Green and IntelligentnTechnology (CIGIT), Chinese Academy of Sciences (CAS), China in 2019 and then joined CIGIT, CAS, China. He is currently a Professor of the College of Computer and Information Science, Southwest University, Chongqing, China. He has over 60 publications, including 14 IEEE Transactions papers of IEEE T-KDE, T-NNLS, T-SC, T-SMC, etc., and conferences of AAAI, WWW, ICDM, IJCAI, etc. He is servicing as an Associate Editor for Neurocomputing and Frontiers in Neurorobotics. Homepage: https://wudi1989.github.io/Homepage/

Dong Yue (SM'08-F'21) received the Ph.D. degree in engineering from the South China University of Technology, Guangzhou, China, in 1995. He is currently a professor and dean of Institute of Advanced Technology, Nanjing University of Posts and Telecommunications and also a Changjiang Professor with the Department of Control Science and Engineering, Huazhong University of Science and Technology. He is currently an Associate Editor of the IEEE Control Systems Society Conference Editorial Board and also an Associate Editor of the IEEE Transactions on Neural Networks and Learning Systems, Journal of The Franklin Institute, International Journal of Systems Science. Up to now, he has published more than 100 papers in international journals. His research interests include analysis and synthesis of networked control systems, multiagent systems, optimal control of power systems, and internet of things.

Xiong Fu received his BS and PhD in Computer Science from the University of Science and Technology of China, Hefei, in 2002 and 2007, respectively. Dr. Xiong Fu is a professor of Computer Science at the Nanjing University of Posts and Telecommunications. He His research interests include parallel and distributed computing, and cloud computing.

Yi He (S'19-M'21) received his Ph.D. degree in computer science from the University of Louisiana at Lafayette and a B.E. from the Harbin Institute of Technology (China). Dr. Yi He is an assistant professor of Computer Science at ODU. His research focus lies broadly in data mining and machine learning and specifically in online learning, data stream analytics, graph learning, recommender systems, and explainable artificial intelligence. His research outcomes have appeared in premier venues, e.g., AAAI, IJCAI, WWW, ICDM, SDM, T.KDE, T.NNLS, among many other AI, machine learning, and data mining outlets.